Infini-T: The Infinite Thread Machine

Krste Asanovic

Computer Science and Artificial Intelligence Laboratory

Massachusetts Institute of Technology

12th SIAM Conference on Parallel Processing for Scientific Computing,

San Francisco, CA

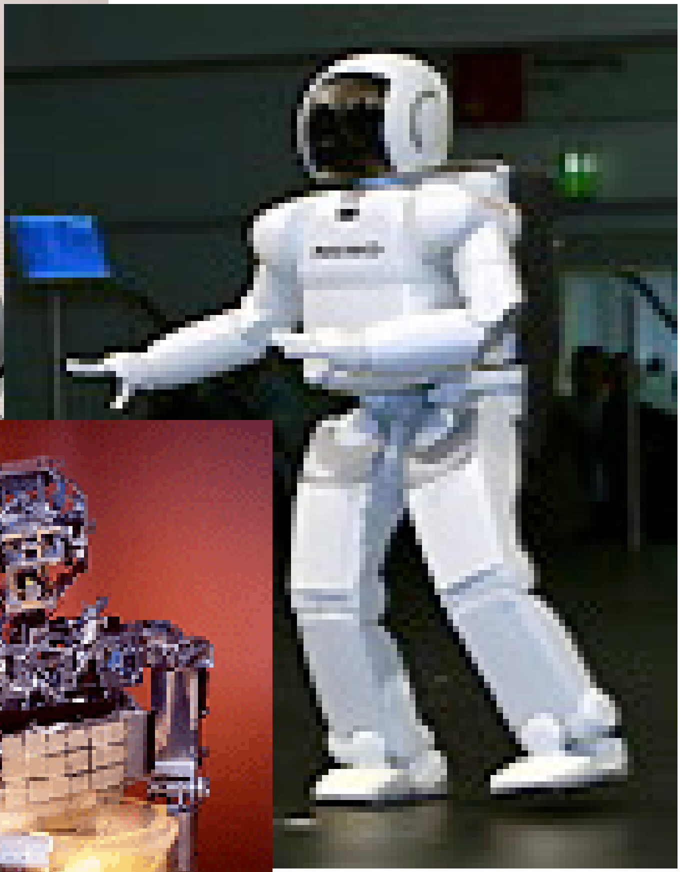
24 February 2006

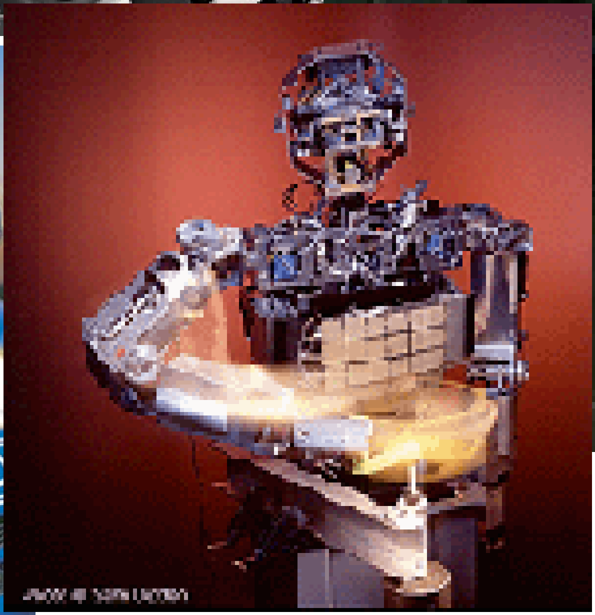
Supercomputing-Driven Architecture?

- Scientific computing can't justify new architectures
  - Market too small for full-custom chip design
  - Compare $1bn/year capability market versus $0.4bn Cell processor development budget
  - Custom-chip design cost rising faster than supercomputing revenue
- Supercomputing systems must reuse mass-market components
  - Processors, DRAMs, FPGAs, network switches, …
- Recent application drivers: media, games, and internet servers
  - Impact on scientific computing: SSE2/3, higher memory and I/O bandwidths, bigger and faster disks

Next Architecture Driver: Robotics?

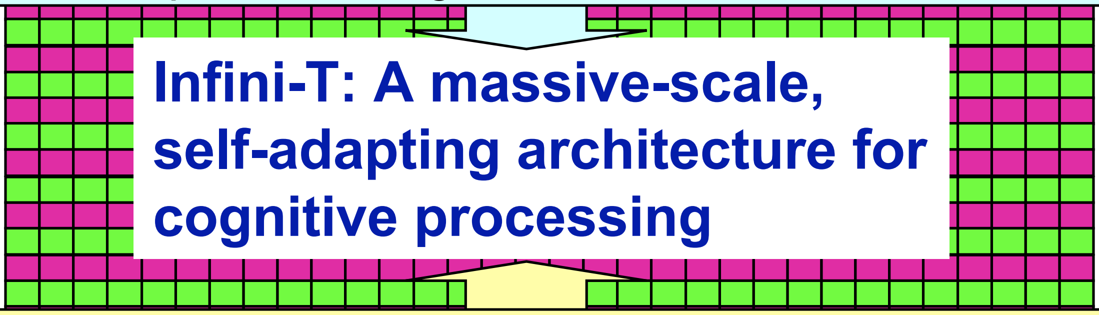
Autonomous robots might lead to robust, adaptive, massively parallel microprocessors for scientific computing





Infini-T Architecture Motivations

- **Cognitive Application Challenges and Opportunities**
  - **Scalability**: Complex irregular processing over large data-sets with aggressive real-time goals requiring massive performance
  - **Adaptivity**: Processing needs vary dynamically and unpredictably requiring automatic reallocation of resources
  - **Resiliency**: Many soft-computing algorithms can tolerate reduced precision, corrupt, or missing values



Infini-T: A massive-scale, self-adapting architecture for cognitive processing

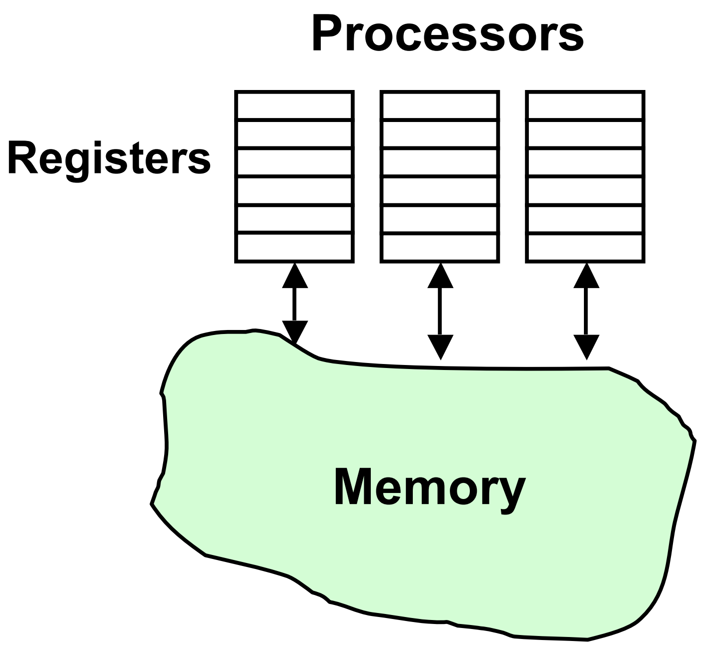
- **Technology Challenges and Opportunities**
  - **Density**: Increased transistor count enables massive on-chip parallelism
  - **Power**: Constraints on switching+leakage power and die temperature require aggressive dynamic power management
  - **Faults**: Increased soft and hard errors require dynamic checking and automatic reconfiguration

Infini-T Key Ideas

- Fine-grain synchronization and context swapping
  - Stored-processor architecture
  - Unbounded hardware transactional memory
  - Producer-consumer synchronization
- Hardware isolation to support self-adaptation
  - Fine-grained Mondriaan memory protection
  - Non-blocking synchronization (transactions)
  - Interconnect and memory bandwidth allocation
  - Power allocation
- No operating system (just a nanokernel)
  - Arbitrarily recursive user-level resource management

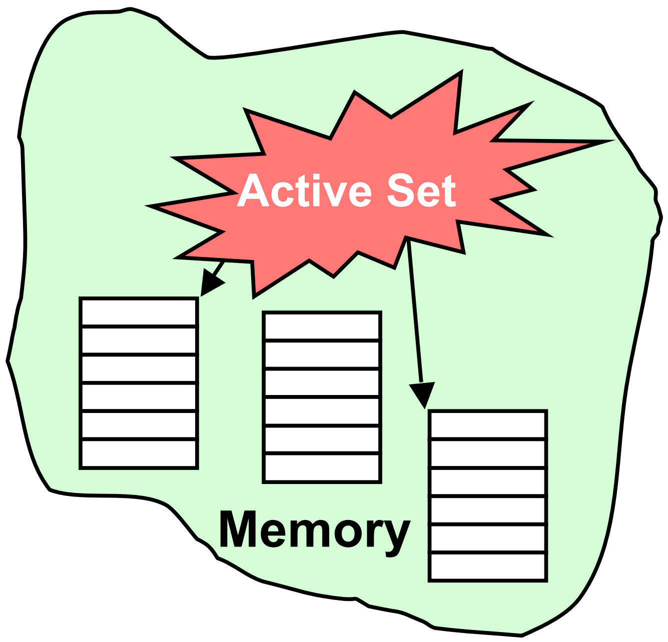
Stored-Processor Computer

Conventional Multiprocessor



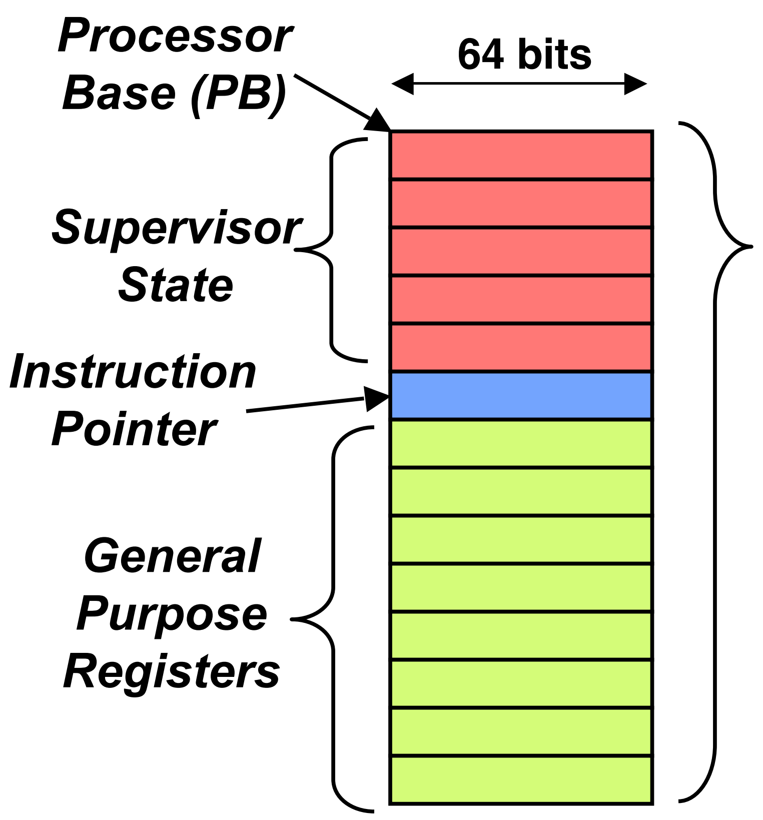
- Software manages exactly N processors
- All processors constantly running
- A processor cannot view or modify another processor's state

Stored-Processor Computer



- Software creates as many processors (hardware threads) as needed
- Only active processors running
- Every processor's state resides in globally accessible memory

Infini-T Processor Programmer's Model



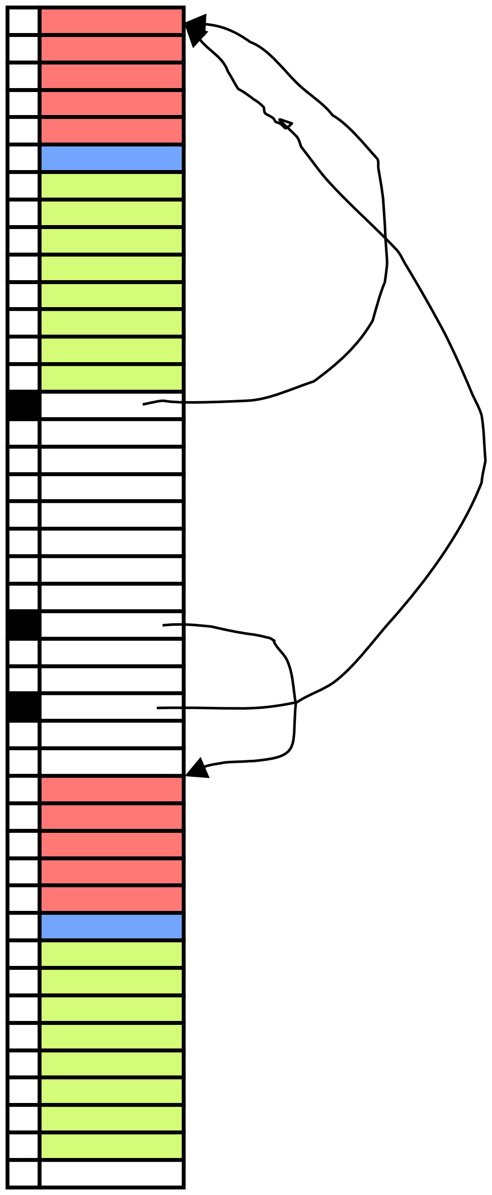
Processor state fits on one or more cache lines (~256B)

Compact register-register instruction encoding maps to memory-memory operations:

ADD R1, R2, R3
=>
M[PB+R1] <- M[PB+R2]+M[PB+R3]

LOAD R1, offset(R2)
=>
M[PB+R1] <- M[M[PB+R2]+offset]

Infini-T Memory Ownership Bits



- Every 64-bit word in memory has an associated ownership bit that indicates the word has been claimed by another processor
- Word holds pointer to current owner
- Owner is responsible for recreating value stored at location
- Used to provide various forms of fine-grained synchronization

Rendezvous Synchronization

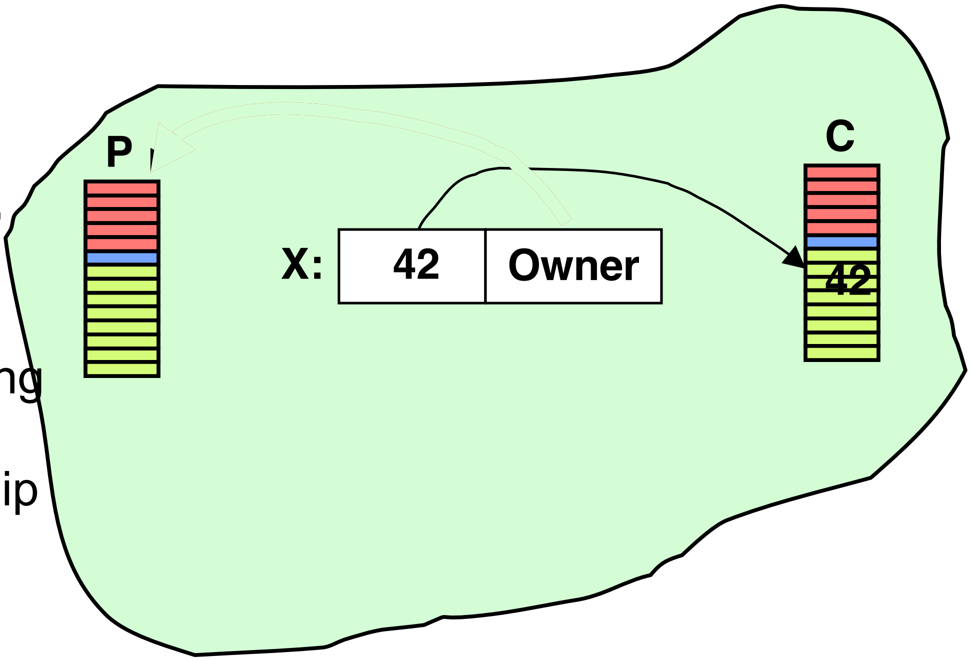
- Producer Processor

```
..
STORE_RNDZV X, 42
..
```

- Consumer Processor

```
..
LOAD_RNDZV X
..
```

- Producer arrives first
- Takes ownership, stores value, and suspends
- Consumer arrives second
- Reads value and wakes sleeping producer
- Producer relinquishes ownership



- If consumer arrives first, it takes ownership and suspends.
- When producer arrives it stores value and wakes consumer.

Synchronizing Streams

- Producer Processor

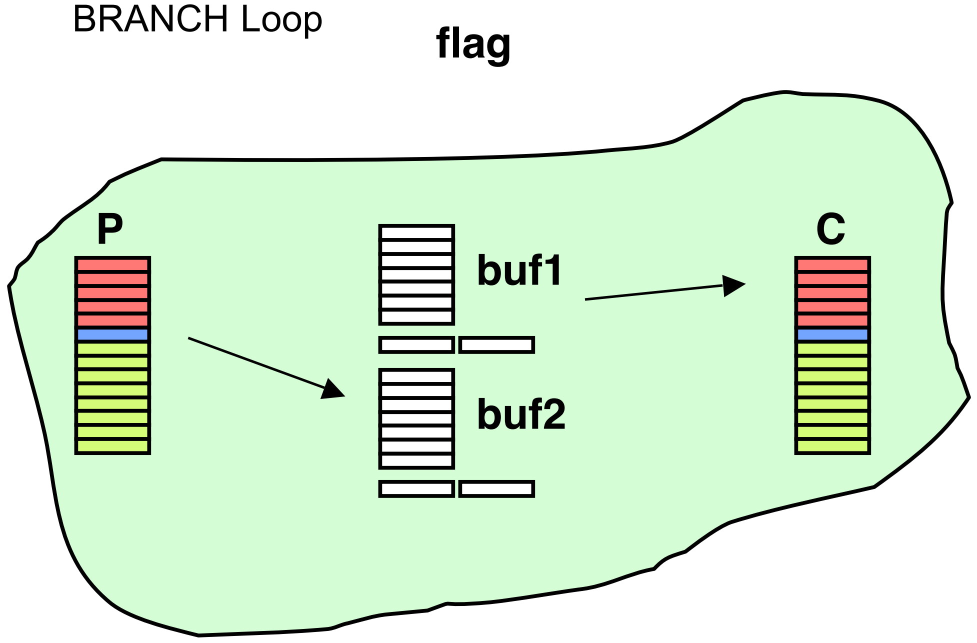
```
Loop:
STORE buf1.data[0] # Fields in
STORE buf1.data[1] # first record.
…
STORE_RNDZV buf1.flag # Done.
..
STORE buf2.data[0] # Fields in
STORE buf2.data[1] # second record.
…
STORE_RNDZV buf2.flag # Done.
..
BRANCH Loop
```

Rendezvous

Rendezvous

- Consumer Processor

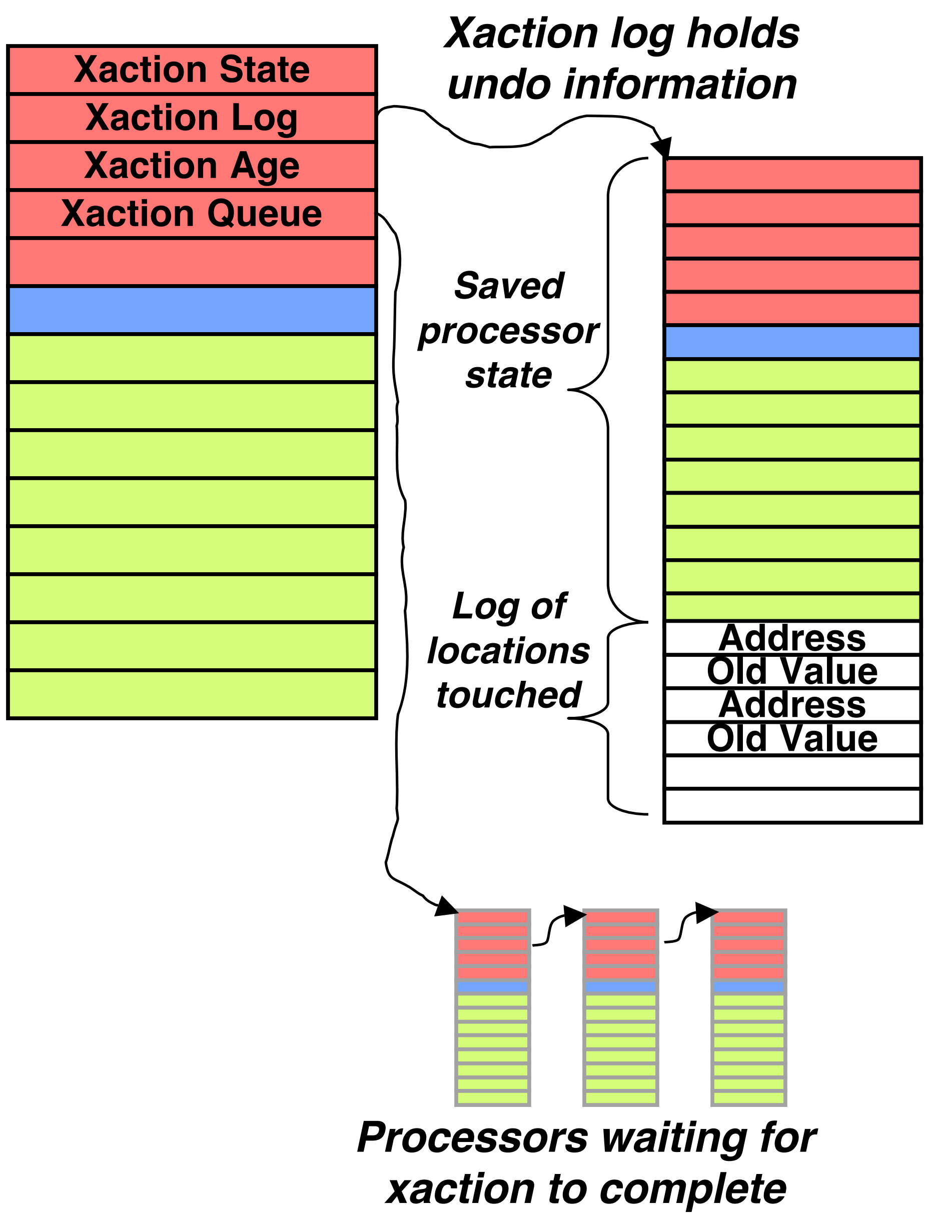
```
Loop:
LOAD_RNDZV buf1.flag # Ready.
..
LOAD buf1.data[0] # Fields in
LOAD buf1.data[1] # first record.
…
LOAD_RNDZV buf2.flag # Ready.
..
LOAD buf2.data[0] # Fields in
LOAD buf2.data[1] # second record.
…
BRANCH Loop
```



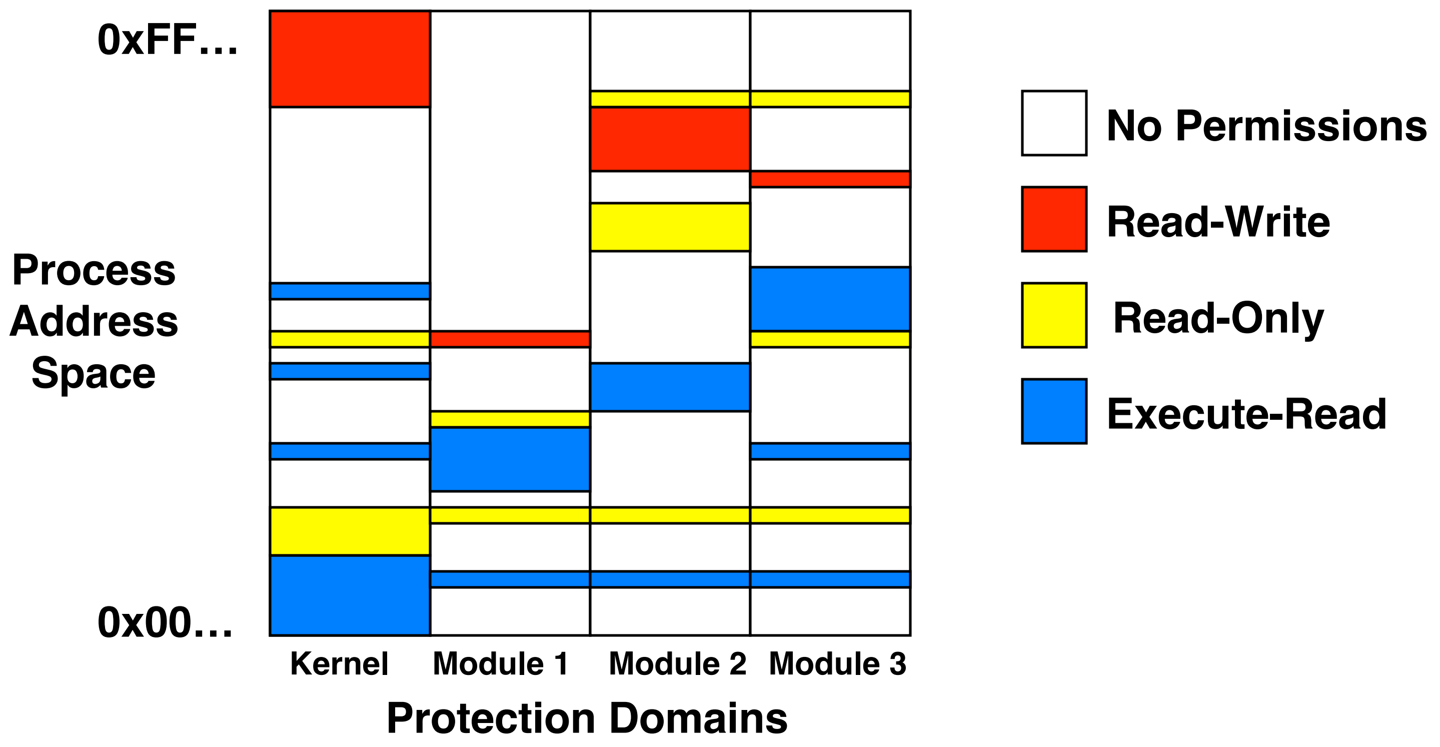
- Example is a double-buffered synchronizing stream.
- Can add additional buffers and threads to provide more decoupling and higher throughput.

Infini-T Transactions

- Infini-T provides "unbounded transactions" *[HPCA'05]*
  - No limit on transaction size or duration
- Instruction XBEGIN/XEND delimit transaction
  - XBEGIN arguments are pointer to log structure and error handler
  - Transaction undone and error handler called if log structure too small
- Each processor tagged with transaction age
  - Set to global clock when processor created
  - Increments on every successful XEND
  - Oldest processor wins on transactional conflict to avoid deadlock and starvation
- Processor state is either:
  - PENDING (running transaction)
  - ABORTED (failed, cleaning up)
  - COMMITTED (successful, cleaning up)



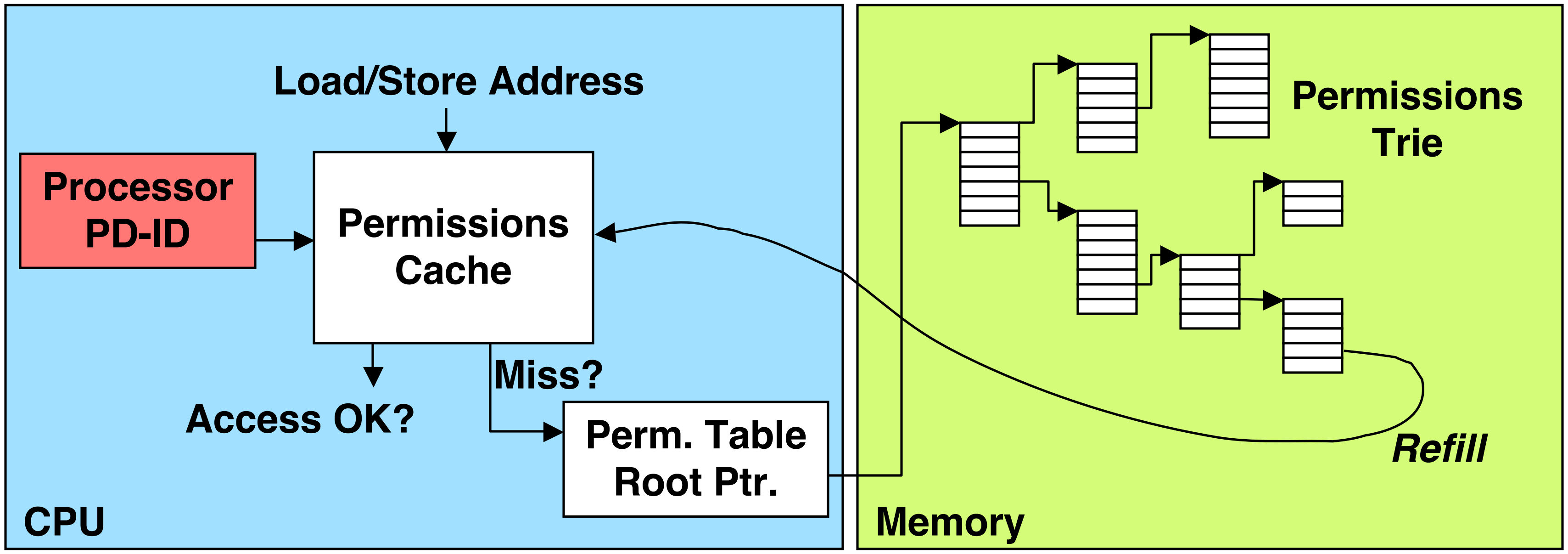
Mondriaan Memory Protection



- Fine-grained word-level memory protection between software modules.
- Enforces module boundaries to give isolation.
- Processor state includes current protection domain identifier (PD-ID).
- Processors can only jump between protection domains at specially marked call and return gates in memory space.

[papers in ASPLOS'02, SOSP'05]

Mondriaan Implementation

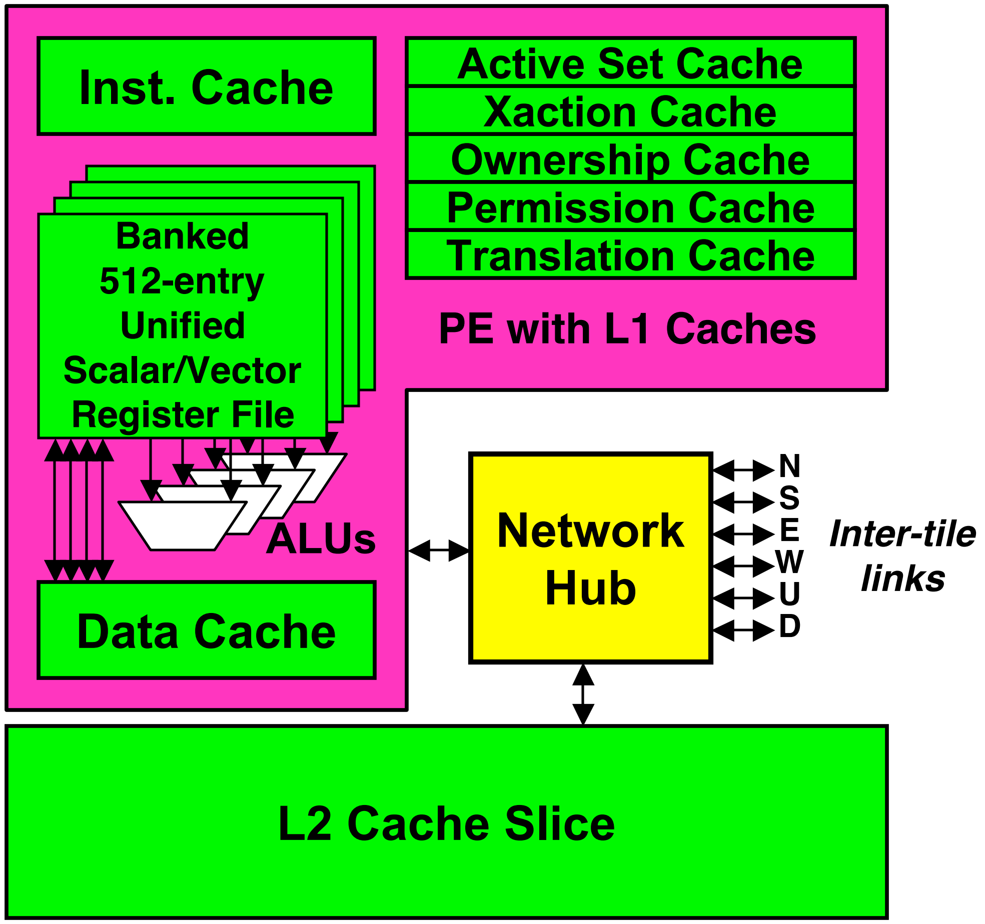


- Efficient, compressed permissions trie structure held in main memory
- Special compressed permissions cache in CPU avoids most permissions lookups in memory structure (<10% overhead)
- Same general structure can be used to associate other metadata with an address or range of addresses

Infini-T Tile

ILP/TLP/DLP execution shares common resources

- Scalar and vector registers allocated out of common register file
- Functional units shared between scalar and vector
  - Multithreaded scalar execution at up to 4x64b instructions per cycle
  - Vector execution at up to
    - 4x64b FLOPS/cycle
    - 8x32b FLOPS/cycle
    - 16x16b OPS/cycle
    - 32x8b OPS/cycle
- High-bandwidth path between register file and primary data cache used for fast thread context save/restore and vector load/store



Infini-T Architecture Overview

Cognitive Application Layer

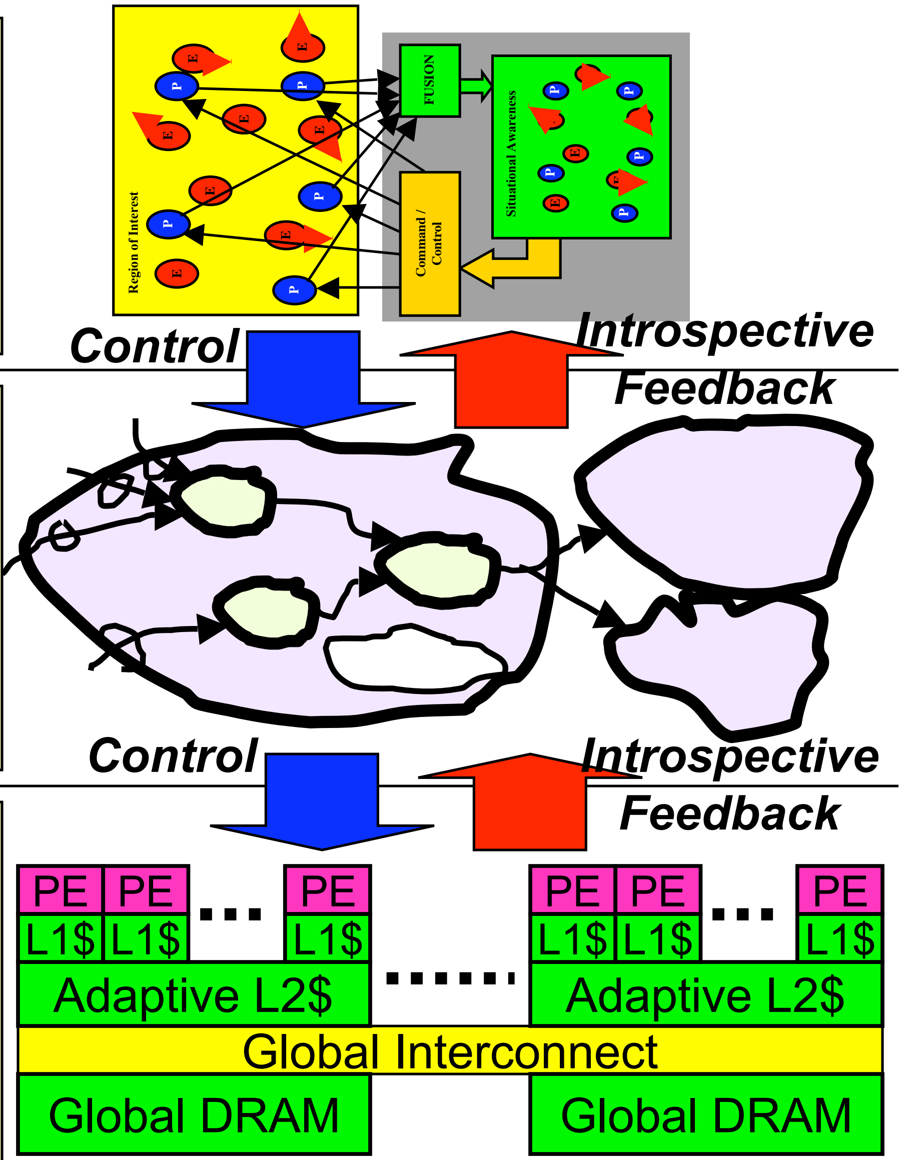
- Goal-based program specification with meta-data (requirements, constraints, hints, etc.)
- Cognitive soft and hard algorithms
  - KB-inference, probabilistic, evolutionary,

Self-Managing Cellular Software

- Knowledge-based compilation and code instrumentation
- Adaptive run-time management
  - Processors, cache policies, locality, interconnect bandwidth, reliability, power, temperature, error recovery

Infini-T Hardware

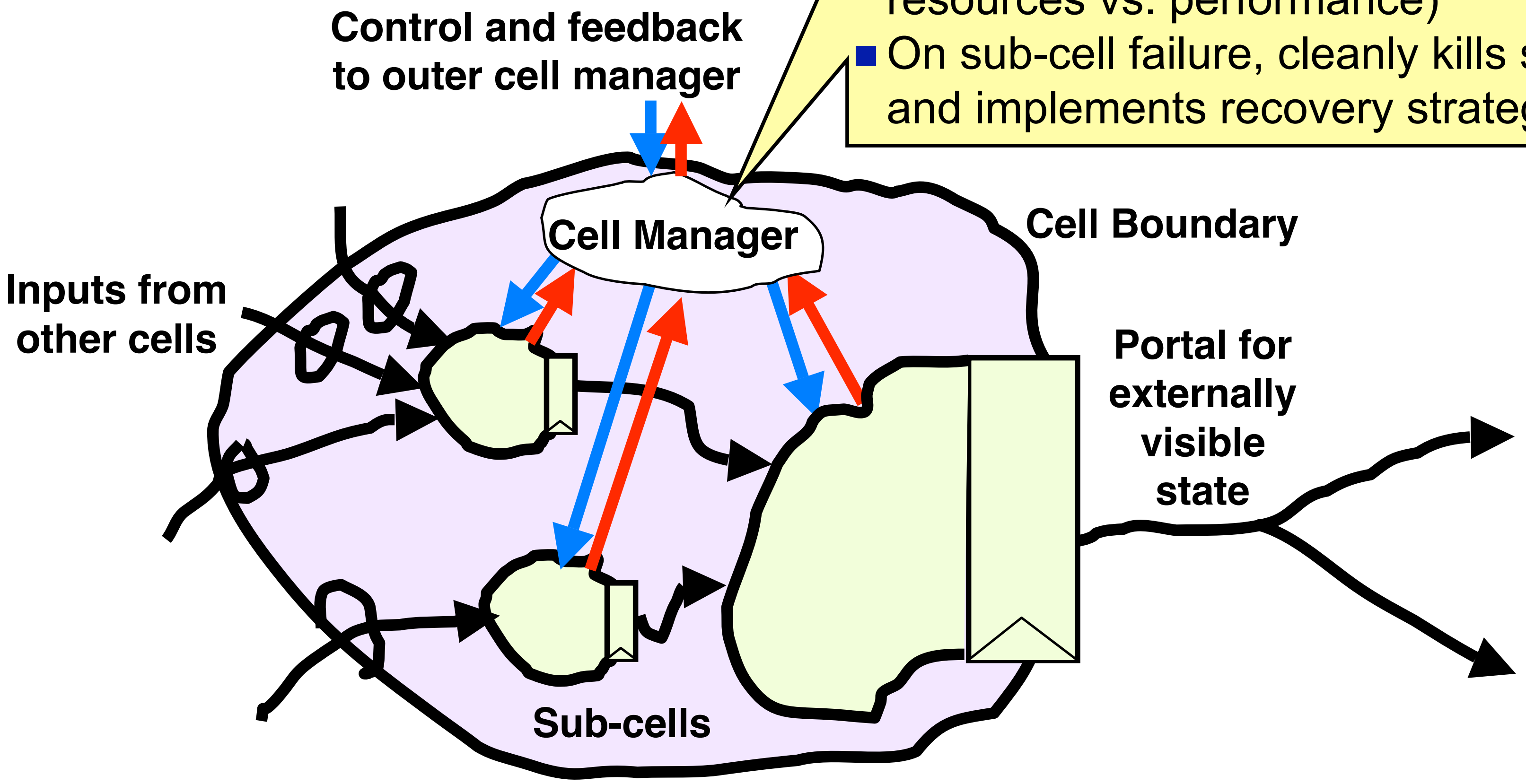
- Massive parallelism
  - 100s cores/chip, 100s chips/system, millions of hardware threads
- Isolation and introspection mechanisms
  - Stored-processors, transactions, memory protection, QoS interconnect



Cellular Run-Time Environment

- Application divides computation into a hierarchical collection of ***cells***
- Each cell is granted resources including processing tiles, memory, global bandwidth, and power.

Cell manager (application-specific code):
- Spawns sub-cells and assigns them resources to run subtasks
- Performs introspection, by monitoring behavior of sub-cells
- Learns behavior of sub-cells (e.g., resources vs. performance)
- On sub-cell failure, cleanly kills sub-cells and implements recovery strategy


Control and feedback to outer cell manager
Cell Manager
Cell Boundary
Inputs from other cells
Portal for externally visible state
Sub-cells

Handling Cell Failure

Many reasons for sub-cell failure:

- **Deadline failure**: Insufficient resources (processors, memory, bandwidth, power, etc.) to finish computation by time required
- **Thermal emergency**: Tile temperature limit exceeded
- **Hardware faults**: Permanent hard fault or transient soft error
- **Bugs**: Application code crashes on this input data

Cell manager must:

- **Detect failure:** is error large enough to require failure recovery?
- **Kill sub-cells**: stop further execution
- **Recover resources**: processors, memory, etc.
- **Implement recovery**: e.g., restart sub-cells

If recovery not possible, cell will report failure to next outer cell.

Cell manager should learn from errors

- e.g., by updating knowledge base on performance versus resources

Infini-T Cell Isolation

System provides strong isolation between cells

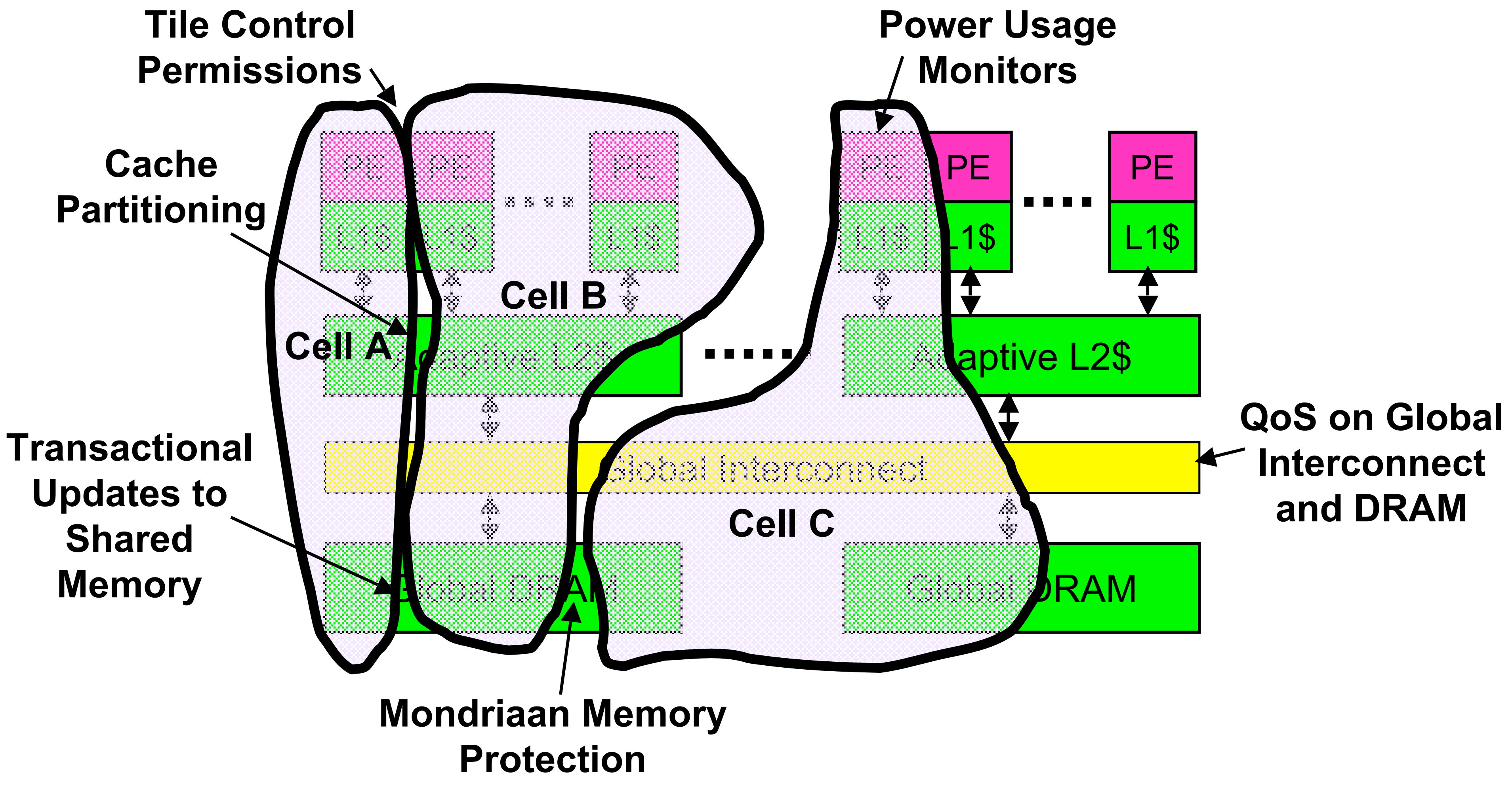
- Limits scope of failure, and simplifies recovery process
- Improves determinism, making it simpler to learn behavior of system for given inputs and given assigned resources

Types of isolation provided in Infini-T

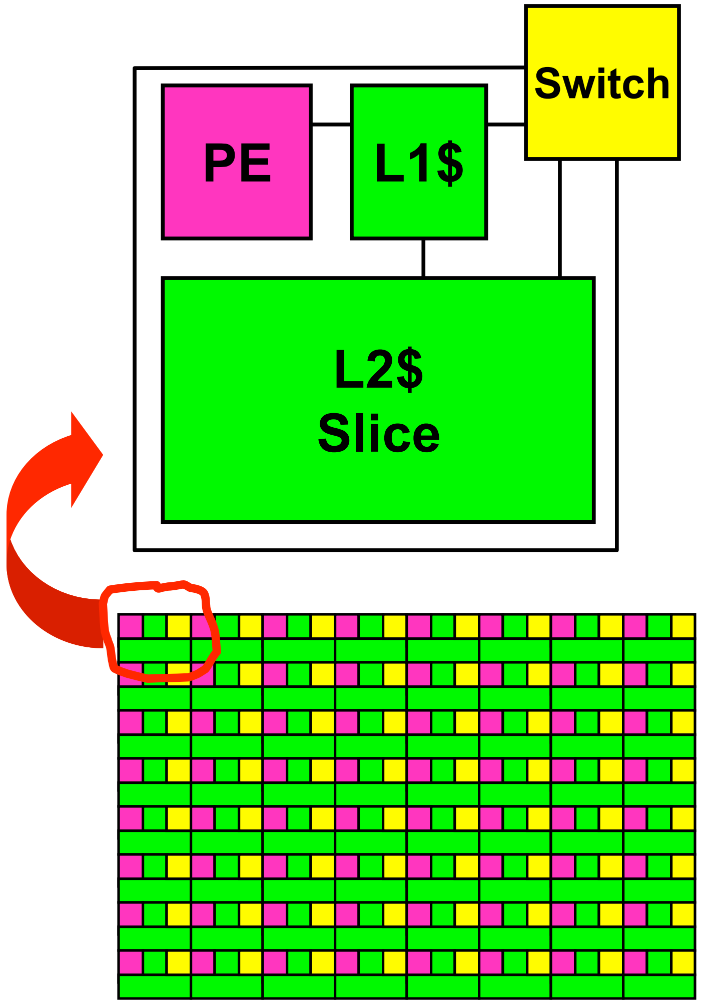
- **Processing cycles:** (tiles*time), limit computational resources used
- **Cache partitioning**: limit cache usage
- **Global memory bandwidth**: limit interconnect BW and DRAM BW
- **Mondriaan Memory Protection**: limit memory accessible to cell
- **Non-blocking synchronization**: avoids cell dying while holding lock
- **Transactions**: avoids cell dying while leaving inconsistent state
- **Power metering:** prevent run-away cell from consuming all power

Hardware Enforces Cell Isolation

…but application cell manager (user-level software) determines policy



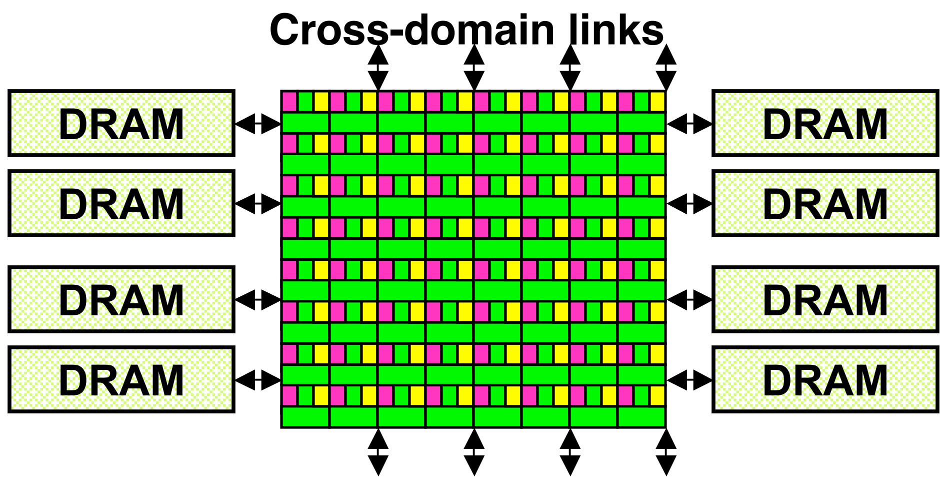
Infini-T Chip-Level Implementation



Physical design organized as replicated tile to reduce design effort and provide redundancy. Each tile contains:

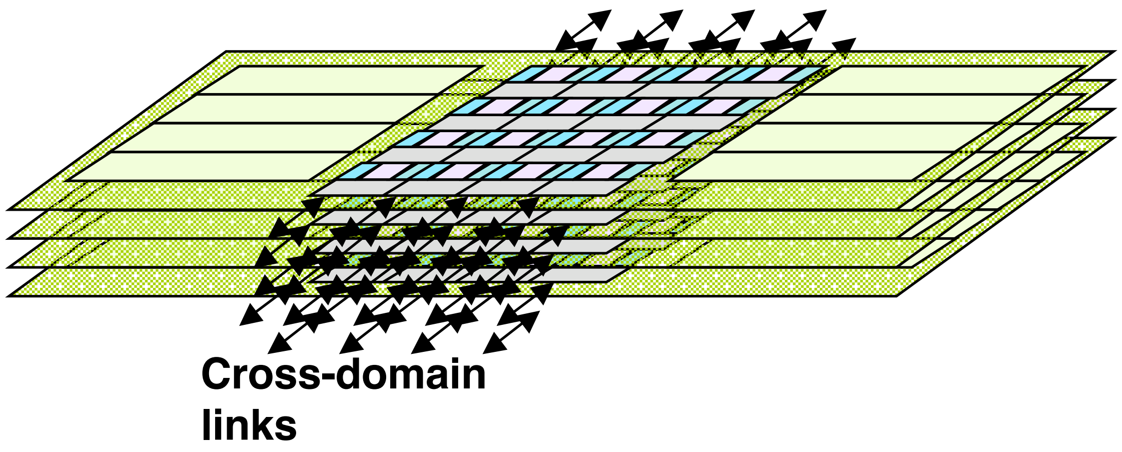
- PE core, with scalar and vector units
- L1 caches (~32KB/tile)
  - Instruction and data caches
  - Processor cache (holds active and sleeping)
  - Translation and permissions caches
  - Active set cache (holds local active threads)
- L2 cache slice (~256KB-1MB/tile)
  - Slices cooperate across all tiles in domain to form large shared NUCA L2
  - Intelligent replication and migration reduces hit latency to L2 resident data
  - L2 level manages coherency across all tile caches in same domain
- Network switch
  - Connects to on-chip network connecting tiles and DRAM & I/O controllers

Infini-T Packaging Options



- Conventional 2D packaging, one Infini-T chip per domain
- Multiple DRAM channels to off-chip DRAM
- Multiple cross-domain connections per chip

- 3D chip stacking, multiple stacked Infini-T chips per domain
- Allows much larger domains, and much higher DRAM bandwidth in each domain
- Multiple DRAM channels per layer
- Multiple cross-domain connections per layer



Summary

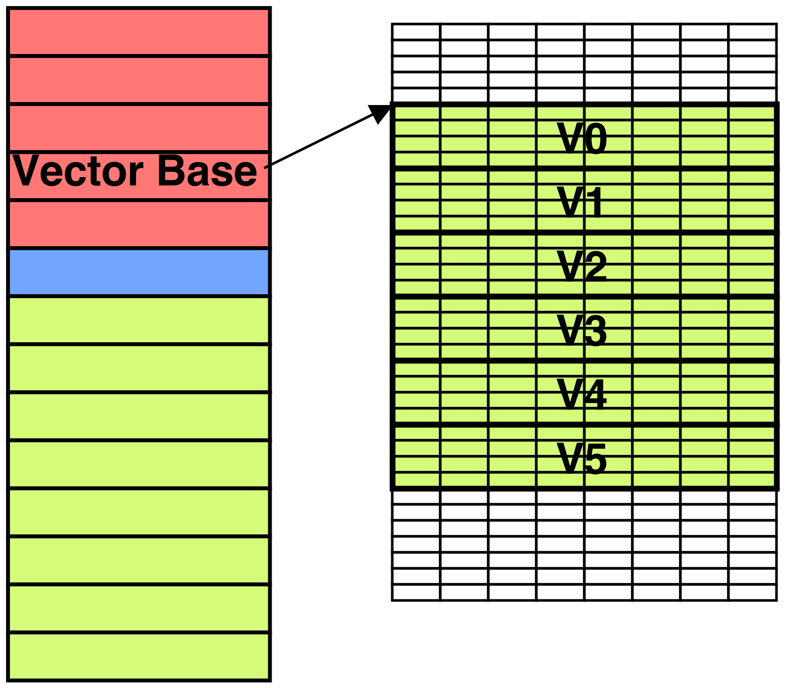
Infini-T is exploring new massively-parallel system architectures that support

- Fine-grained synchronization and context switching
- Hardware isolation and fine-grained protection
- User-level self-management

- *Research sponsored in part by the Defense Advanced Research Projects Agency (DARPA) under the ACIP program and grant number NBCH104009*

Backup Slides

Infini-T Vector Support



```
VCONFIG R1, #6     ; Allocate space
VLD V1, R2         ; Regular encoding
VLD V2, R3
VADD V3, V1, V2
VST V3, R4
```

- Vector registers held in memory on contiguous cache lines
  - Up to 32 vector registers, each 32 elements of 64 bits
- Software must configure vector unit with base address and required number of vector registers before use
- Vector registers cached in vector unit during operation, full chaining supported
- Full support for unit-stride, strided, scatter-gather
  - Effectively become memory-memory copies
- Support for narrower width vector operands
  - 32x64b, 64x32b, 128x16b, 256x8b

Infini-T Implementation

- Entire state of parallel program execution is visible as a memory-resident data structure
  - Supports introspection and debugging.
- Special-purpose caches avoid most memory traffic for common operations
  - Transaction logs only actually created in memory if transaction large and/or contested.
- Mondriaan memory protection scheme restricts access to system data structures
  - Processor supervisor state, protection tables, ownership tables.
- Each Infini-T instruction requires a bounded small number of memory locations to be updated atomically.
  - Underlying coherence protocol supports small transactions.

Infini-T Transaction Execution

- XBEGIN copies initial processor state to log, sets processor Xaction state to PENDING
- Loads claim ownership and record address
- Stores claim ownership, record address and old value, and update memory
- If XEND reached without conflict, processor switches to COMMITED state and begins revoking ownership on locations
- On conflict between two PENDING transactions, oldest processor wins.
  - Losing processor placed on waiting queue of winning processor (No point wasting cycles running losing processor until winner completes)
  - Losing processor enters ABORTED state (while waiting on queue) and begins revoking ownership and restoring old values.
  - Winning processor continues running once contested location restored.
- If PENDING transaction encounters owner that is COMMITED or ABORTED, then places self on owner's queue to await clean up.
  - Optimization is to force cleanup of contested location immediately.
- When processor finishes committing or aborting, it wakes up any queued processors

Exploiting Soft Computing

- Technology scaling will lead to chips with many errors:
  - Soft errors from particle strikes, worst-case coupling noise, power supply glitches, borderline fabrication quality
  - Hard errors from reliability failures over time, burn-in less effective in finer technologies

….but Soft Computing can sometimes tolerate errors

- Three levels at which errors can be reported or exploited:
  - Application code/cell level - managed by dynamic software system
  - Thread/transaction level - managed by dynamic hardware system
  - Instruction level - managed statically by compiler
- Can use reduced supply voltage, or increased clock rate, or less error correction circuitry, and tolerate resulting errors
- Cell isolation vital to detect and recover from lethal errors in cell (i.e. don’t have to guarantee all errors are benign)

Instruction-Level Error Resilience

- Approximate / probabilistic data can be corrupted without harm
- Not all instructions in soft computations are resilient to error
  - Instructions along data/control flow to approximate data are *potentially* resilient
  - Which ones are resilient?

3 Major Instruction Types

Arithmetic Instructions
- All of these are resilient
- Safety: ignore exceptions and return a “reasonable” value

Memory Instructions
- Bad load or store data is o.k.
- Bad load address is o.k.; on exception, return a reasonable value
- Bad store address is not o.k. (but caught by Mondriaan)

Control Instructions
- Bad direction is o.k. in some cases: early loop exit, if-then-else
- Bad target address is not o.k. (but caught by Mondriaan)

CEARCH Architecture Prototype

- InfiniT chip
  - ≥ 128 cores per chip
  - ≥ 1M light-weight threads per chip
- CEARCH system
  - ≥ 64 chips per system
  - Transactional memory
  - Adaptive hardware and runtime